

**PATENT APPLICATION
DOCKET NO. 0100.9901080**

In the United States Patent and Trademark Office

FILING OF A UNITED STATES PATENT APPLICATION

Title:

**METHOD AND APPARATUS FOR DYNAMIC ALLOCATION OF
PROCESSING RESOURCES**

Inventors:

Korbin Van Dyke 3343 Little Valley Road Sunol, CA 94586 USA	Paul Campbell 6652 Dana Street Oakland, CA 94609 New Zealand
Don Van Dyke	Ali Alasti 1566 Siesta Drive Los Altos, CA 95024 USA
Stephen C. Purcell 365 Preston Drive Mountain View, CA USA	

**Attorney of Record
Ross D. Snyder
Registration No. 37,730
175 W. Jackson Blvd. – Suite 1015
Chicago, Illinois 60604
Phone (512) 894-3496
Fax (512) 858-1105**

Express Mail Label No. EL454270259US

Date of Deposit: 1-18-00
I hereby certify that this paper is being deposited
with the U.S. Postal Service "Express Mail Post
Office to Addresses" service under 37 C.F.R. Section
1.10 on the 'Date of Deposit', indicated above, and is
addressed to the Commissioner of Patents and
Trademarks, Washington, D.C. 20231.

Name of Depositor: Kris Herzog
(print or type)

Signature: Kris Herzog

091484549

METHOD AND APPARATUS FOR DYNAMIC ALLOCATION OF
PROCESSING RESOURCES

Technical Field of the Invention

The invention relates generally to computer architectures and more specifically to computer architectures for processing multimedia data.

Background of the Invention

Computers traditionally developed around a single central processing unit. As more differ ways of getting information to and from the central processing unit were developed, they were attached to the central processing unit, but the central processing unit remained essentially unchanged.

FIG. 1 is a block diagram illustrating an example of a prior art computer architecture. Traditional data processing systems comprise a general-purpose processor coupled to a variety of application-specific subsystems. Processor 101 is coupled to an application-specific communications subsystem 102, to an application-specific audio subsystem 103, and to an application-specific video subsystem 104.

Such prior art computer architectures suffer from inflexibility in the allocation of resources. For example, when performing processing that does not involve processing of audio data, audio subsystem 103 remains unused. It cannot be used for other purposes since it is permanently configured specifically for processing audio data. Thus, the overall processing capability of the computer architecture is limited by the processing capability of processor 101.

Attempts in the prior art to avoid the inefficiency of the separate application-specific communications subsystem 102, application-specific audio subsystem 103, and application-specific video subsystem 104 have used a single combined subsystem to perform the functions of subsystems 102, 103, and 104. However, such single combined subsystems still suffer from inflexibility with respect to allocation of resources between the single combined subsystem and processor 101.

Likewise, the processing capability of processor 101 is also limited. For example, a typical modern microprocessor, such as a Pentium III, often executes 200-300 cycle instructions that cannot be interrupted. Also, a typical operating system for such a microprocessor, such as Microsoft Windows, can have high and unpredictable latency with limitations, such as floppy disk accesses that cannot be interrupted.

Therefore, a method and apparatus is needed to provide increased flexibility and efficiency in the allocation of resources in a computer system.

Brief Description of the Drawings

FIG. 1 is a block diagram illustrating an example of a prior art computer architecture.

FIG. 2 is a block diagram illustrating a processing apparatus according to an embodiment of the invention.

FIG. 3 is a block diagram illustrating a processing apparatus according to an embodiment of the invention.

FIG. 4 is a block diagram illustrating kernel functionality according to an embodiment of the invention.

FIG. 5 is a timing diagram illustrating allocation of tasks among processing resources over time according to an embodiment of the invention.

FIG. 6 is a block diagram illustrating layers of processing functionality in a prior art computer system.

FIG. 7 is a block diagram illustrating layers of processing functionality according to an embodiment of the invention.

FIG. 8 is a flow diagram illustrating a process for providing multimedia functionality in a homogeneous multiprocessor environment according to an embodiment of the invention.

FIG. 9 is a block diagram illustrating a structure capable of supporting software applications and processing multimedia data according to an embodiment of the invention.

FIG. 10 is a flow diagram illustrating a process for passing resulting data to a first input/output device according to an embodiment of the invention.

Detailed Description of a Preferred Embodiment

The invention provides a method and apparatus for dynamic allocation of processing resources. By using a plurality of processors designed to have low and predictable interrupt latency that is capable of being configured to efficiently support multimedia processing without being encumbered by the Windows operating system while also being capable of being configured to efficiently support the Windows operating system, one embodiment of the invention increases the flexibility and efficiency of allocating system resources.

In accordance with one embodiment of the invention, a single integrated circuit design may be provided to support both general-purpose processing and multimedia processing. This obviates the need to design a separate device for each function to be performed by a computer system. Several instances of the single integrated circuit design may be provided in a system to increase the capability of the system. Processing resources can be efficiently allocated among tasks. Changes to the configuration of the system can be effected by changes in software, without the need for replacement of or addition to hardware.

FIG. 2 is a block diagram illustrating a processing apparatus according to an embodiment of the invention. General-purpose processors 202, 203, 204, and 205 are substantially identical and are coupled to bus 201. Each of processors 202, 203, 204, and 205 may be programmed to function as general-purpose processors or to provide the same functionality as found in an application-specific subsystem. Processors 202, 203, 204, and 205 may communicate with each other and with other components and subsystems via bus 201.

FIG. 3 is a block diagram illustrating a processing apparatus according to an embodiment of the invention. As described in reference to FIG. 2, processors 202, 203, 204, and 205 are coupled to bus 201. Memory interface 306 is also coupled to bus 201. Memory device 307 is coupled to memory interface 306.

Input/output interface 308 is coupled to bus 201. A first input/output device 309, which might, for example, be a video device, is coupled to input/output interface 308. A second input/output device 310, which might, for example, be an audio device, is coupled to input/output interface 308. A third input/output device 311, which might, for example, be a CD or DVD disk device, is coupled to input/output interface 308.

Processors 202, 203, 204, and 205 may be provided with a set of instructions in a program to cause them to perform different functions. For example, processor 202 may function as a general-purpose processor, while processor 203 may be provided with instructions to cause it to function as a video processor. As a video processor, processor 203 can communicate with a video device, for example, first input/output device 309 via bus 201 and input/output interface 308.

As another example, processor 204 may be provided with instructions to cause it to function as an audio processor. As an audio processor, processor 204 can communicate with an audio device, for example, second input/output device 310 via bus 201 and input/output interface 308.

Likewise, as another example, processor 205 may be provided with a set of instructions in a program to cause it to function as a CD or DVD processor.. As such, processor 205 can communicate with a CD or DVD device, for example, third input/output device 311 via bus 201 and input/output interface 308. U.S. Patent Application No. 09/427,168, filed October 26, 1999, entitled *Transferring Execution from One Computer Instruction Stream to Another*, is incorporated herein by reference.

Processors 202, 203, 204, and 205 can perform memory operations on memory device 307 via bus 201 and memory interface 306. While four processors are illustrated, the invention may be practiced with any number of processors.

The system may be initialized by having one of processors 202, 203, 204, or 205 obtain instructions from a known memory location. The instructions determine how the processors should be configured initially. The system may be subsequently reconfigured if desired.

FIG. 4 is a block diagram illustrating kernel functionality according to an embodiment of the invention. A kernel 407 provides the ability to allocate tasks 404, 405, and 406 among processing resources 401, 402, and 403. Tasks 404, 405, and 406 represent any processes or operations which require some amount of processing. Constraints, priorities, and dependencies may exist between tasks 404, 405, and 406. For example, a task may need to be completed by a particular time or before or after another task, and some tasks may be more important than others.

Processing resources 401, 402, and 403 represent capability for processing tasks 404, 405, and 406. This capability may include processors 202, 203, 204, and 205. A portion of processors 202, 203, 204, and 205 may be reserved for other uses and not included within processing resources 401, 402, and 403.

Kernel 407 quantifies the capability of processing resources 401, 402, and 403, quantifies the processing requirements of tasks 404, 405, and 406 subject to any constraints, priorities, and dependencies that may exist, and dynamically allocates tasks 404, 405, and 406 to processing resources 401, 402, and 403.

Thus, kernel 407 according to one embodiment of the invention can be considered to provide a method for dynamically allocating processing resources for among a plurality of tasks, for example including multimedia tasks, or a method for dynamically allocating a plurality of tasks to a plurality of processors, which may be identical processors.

Kernel 407 may include portions expressed in different instruction sets, for example, a native instruction set and an x86 instruction set. These different instruction

sets may include a first instruction in a first instruction set that is denoted by a particular bit pattern and a second instruction in a second instruction set that is denoted by the same particular bit pattern. However, the first instruction in the first instruction set may denote a different operation to be performed than the operation to be performed as denoted by the second instruction of the second instruction set.

Kernel 407 may coordinate different tasks expressed in different instruction sets. For example, task 404 may be expressed in a native instruction set, while task 405 may be expressed in an x86 instruction set. Kernel 407 allows resulting data that results from the performance of one task expressed in one instruction set to be used as initial data in the performance of another task expressed in another instruction set. U.S. Patent Application No. 09/239,194, filed January 28, 1999, entitled *Executing Programs for a First Computer Architecture on a Computer of a Second Architecture*, is incorporated herein by reference.

FIG. 5. is a timing diagram illustrating allocation of tasks among processing resources over time according to an embodiment of the invention. The diagram shows how processing resources and tasks may be divided in time and space. For example, a first process 501 is allocated to a first processing resource P_1 . The first process 501 is completed by the first processing resource P_1 at time T_1 . At the same time, a second process 502 is allocated to a second processing resource P_2 . The second process 502 is completed by the second processing resource P_2 at time T_2 . At the same time that the first process 501 and the second process 502 are being processed, a third process 503 is being processed by a third processing resource P_3 . The third process 503 is completed by the third processing resource P_3 at time T_3 .

After time T_1 , the first processing resource P_1 is available to be allocated for other tasks. At time T_4 , the first processing resource P_1 is allocated to begin processing of a fourth process 504. At time T_2 , the second processing resource P_2 becomes available to be allocated for other tasks and is allocated to assist in the processing of the fourth process 504. At time T_3 , the third processing resource P_3 becomes available to be

allocated for other tasks and is allocated to assist in the processing of the fourth process 504. Thus, after time T_3 , first processing resource P_1 , second processing resource P_2 , and third processing resource P_3 are all allocated to processing fourth process 504.

The invention may also be used to divide the processing capability of a single processor between multiple tasks. Thus, if two tasks are being performed in a system with three processors, the first processor could be assigned to perform the first task, the second processor could be assigned to perform the second task, and the processing capability of the third processor could be divided to assist the first processor in performing the first task, and the second processor in performing the second task.

Processing resources may be obtained by dividing or aggregating the processing capability of one or more processors. For example, the processing capability of a single processor may be divided in space or time to provide a processing resource. Dividing the processing capability in space refers to using some amount of the processing capability of a processor to provide one processing resource while simultaneously using other amounts of the processing capability of the processor to provide other processing resources. Dividing the processing capability in time refers to using the processing capability of a processor for some period of time to provide one processing resource and using the processing capability of the processor for a different period of time to provide another processing resource. The periods of time may be alternated or cycled between processing resources. Also, division of processing capability in space and time are not mutually exclusive, so processing capabilities may be divided in both space and time to provide processing resources.

Processing resources may be obtained by combining the processing capabilities of more than one processor. For example, the processing capabilities of two processors may be combined to provide a processing resource. The processing capabilities of multiple processors need not be combined in integer quantities. Rather, the processing capability of a processor may be divided in space or time and combined with the processing capability, or portions thereof, of another processor to obtain processing resources. The

ability of processing resources to be obtained by dividing and/or aggregating the processing capabilities of more than one processor allow a portion of a plurality of processors to be used for one purpose and another portion of the plurality of processors to be used for another purpose. The portion of the plurality of processors may include processing capability of the plurality of processors combined and/or divided in space and/or time.

FIG. 6 is a block diagram illustrating layers of processing functionality in a prior art computer system. In a traditional computer system, applications 603 are executed on an operating system 602, which, in turn, is executed on an x86 microprocessor 601. The internal architecture of the x86 microprocessor evolved before the processing of multimedia data and, therefore, does not provide substantial native support for input/output capability as is needed for the processing of multimedia data.

Moreover, the internal architecture of the x86 microprocessor is not optimized for efficient processing of multimedia data. Therefore, additional application-specific circuits are needed in conjunction with the x86 microprocessor. These application-specific circuits cannot be adapted to provide the same functionality as the x86 microprocessor. Thus, the x86 microprocessor and the application-specific circuits are immutably configured to perform their pre-determined roles, their processing capabilities cannot be adjusted or re-allocated.

FIG. 7 is a block diagram illustrating layers of processing functionality according to an embodiment of the invention. Either an input/output kernel 703 or an x86 kernel 704 is executed on a processor 705, which need not be an x86 microprocessor. Processor 705 contains structures capable of efficiently supporting operation to provide the functionality of either a microprocessor, such as an x86 microprocessor, or an application-specific circuit, such as circuits for supporting a video, audio, or CD/DVD subsystem. When the x86 kernel 704 is executed on processor 705, the x86 kernel 704 and the processor 705, taken together, support the same instruction set as an x86

microprocessor and may be used in place thereof. Applications 701 are executed on an operating system 702, which is executed on x86 kernel 704 running on processor 705.

When the input/output kernel 703 is executed on processor 705, the input/output kernel 703 and the processor 705, taken together, provide the same functionality as application-specific circuits, such as circuits for supporting a video, audio, or CD/DVD subsystem and may be used instead of such circuits. Thus, depending on whether input/output kernel 703 or x86 kernel 704 is selected, processor 705 may be dynamically assigned to function in place of a general-purpose microprocessor or application-specific circuits.

The input/output kernel 703 or x86 kernel 704 keeps track of available processing resources and can instruct a processing resource to start executing code at a specific address. The kernels 703 and 704 can also utilize a look-ahead capability to forecast and meet task deadlines associated with particular tasks.

Since x86 kernel 704 operates below operating system 702, x86 kernel 704 can manipulate memory used by operating system 702. For example, x86 kernel 704 can take a portion of memory that would otherwise be allocated to operating system 702 and reserve that portion of memory. Likewise, input/output kernel 703 can control or adjust memory allocation.

FIG. 8 is a flow diagram illustrating a process for providing multimedia functionality in a homogeneous multiprocessor environment according to an embodiment of the invention. The homogeneous multiprocessor environment comprises a plurality of substantially identical processors. The plurality of processors are capable of executing a first instruction of a first instruction set and a second instruction of a second instruction set. One manner in which the first instruction set may differ from the second instruction set is that the first instruction of the first instruction set and the second instruction of the second instruction set may share an identical bit pattern but perform different operations. The process begins in step 801. In step 802, the process queues tasks. These tasks may

comprise various types of processing undertaken by a computer system, such as x86 processing, graphic image processing, video processing, audio processing, communication processing, etc. In step 803, the process identifies available processing resources in the homogeneous multiprocessor environment. In step 804, the process allocates the available processing resources among the tasks. This allocation may be dynamically adjusted. In step 805, the process executes in a first processor of the plurality of processors an input/output kernel program. The input/output kernel program includes a first portion expressed using the first instruction set and a second portion expressed using the second instruction set. In step 806, the process receives initial data from an input/output device. In step 807, the process provides functional programs and initial data corresponding to the tasks to the available processing resources. In step 808, the process converts a functional program expressed using a first instruction set to an equivalent functional program expressed using a second instruction set. In step 809, the process performs the tasks using the available processing resources to produce resulting data. In step 810, the process passes the resulting data to a first input/output device. This step may further comprise the step of passing the resulting data through an intermediary device, wherein the intermediary device is coupled to the first input/output device and to a second input/output device, which may further comprise the step of automatically adapting to a reallocation of the available processing resources among the tasks. Alternatively, step 810 may further comprise the step of passing the resulting data to a mixed-signal device. The process ends in step 811.

The invention may be practiced using apparatus comprising a plurality of processors coupled to a bus, an input/output interface coupled to the bus, and a plurality of input/output devices coupled to the input/output interface. The plurality of processors process program code configured to perform a plurality of tasks. The program code comprises program code configured to cause a first portion of the plurality of processors to interact with a first input/output device of the plurality of input/output devices. The program code also comprises program code configured to cause a second portion of the plurality of processors to interact with a second input/output device of the plurality of input/output devices. The program code further comprises program code configured to

cause a second portion of the plurality of processors to emulate a specific microprocessor instruction set. The apparatus may further comprise kernel program code configured to dynamically allocate the processing of the program code among the plurality of processors.

It can be readily appreciated that the input/output devices may be of a bidirectional nature. For example, processing resources may receive initial data from an input/output device, perform tasks to produce resulting data, and pass the resulting data to the same or a different input/output device.

FIG. 10 is a flow diagram illustrating a process for passing resulting data to a first input/output device according to an embodiment of the invention. In FIG. 10, the process begins in step 1001 and continues in either step 1002 or step 1004. At step 1002, the process passes the resulting data through an intermediary device. In step 1003, which follows step 1002, the process automatically adapts to a reallocation of the available processing resources among the tasks. In step 1004, the process passes the resulting data to a mixed-signal device. In step 1005, which follows either step 1003 or step 1004, the process ends.

FIG. 9 is a block diagram illustrating a structure capable of supporting software applications and processing multimedia data according to an embodiment of the invention. Some software applications, for example disk operating system (DOS) programs 901, are supported by virtual device drivers 902, which are programmed in native code. The virtual device drivers 902 are supported by a kernel host library 903.

Other software applications, for example Windows applications 904 are supported by Windows dynamic link libraries 905, which, in turn, are supported by Windows drivers 906, which are programmed in native code. Alternatively, Windows applications 904 may bypass the Windows dynamic link libraries 905 and be supported directly by Windows drivers 906. Windows drivers 906 are supported by kernel host library 903.

[illegible]

13